ISSN 0265-2919

80p 19

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- **Although expensively-priced at £600, the Adam is sold as a complete system. This system includes a high quality daisy wheel printer, word processing software, a separate keyboard, a high speed digital cassette unit, a pair of joysticks and a Colecovision games unit.**
- **We begin a new programming project for the BBC Micro and the Electron**

QUIZ

1) How does bi-directional printing improve the speed of a printer?

2) Which company markets Gridrunner on the Sinclair Spectrum - Llamasoft, Ultimate or Salamander?

3) Why does the Torch Disk Pack come with an additional Z80 processor?

Answers To Last Week's Quiz

A1) 2 (on or off state) to the power of 24 (no. of bits) equals 16777216 (16 Mbytes).

A2) The LCD has a slower response time and is harder to read than a cathode ray tube but is less bulky and uses less power.

A3) An attribute is a command that is not printed directly on to the paper, but formats the print.

QUIZ

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH

Editor Jim Lennox; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Sub Editor** Steve Mann; **Contributors** Geoff Bains, Chris Bidmead, Steve Colwill, Steve Darroch, Guy Lancaster, Max Phillips, Matt Nicholson, Geoff Nairn, Graham Storrs, Richard Pawson; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd; Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Controller** Peter Taylor-Medhurst; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

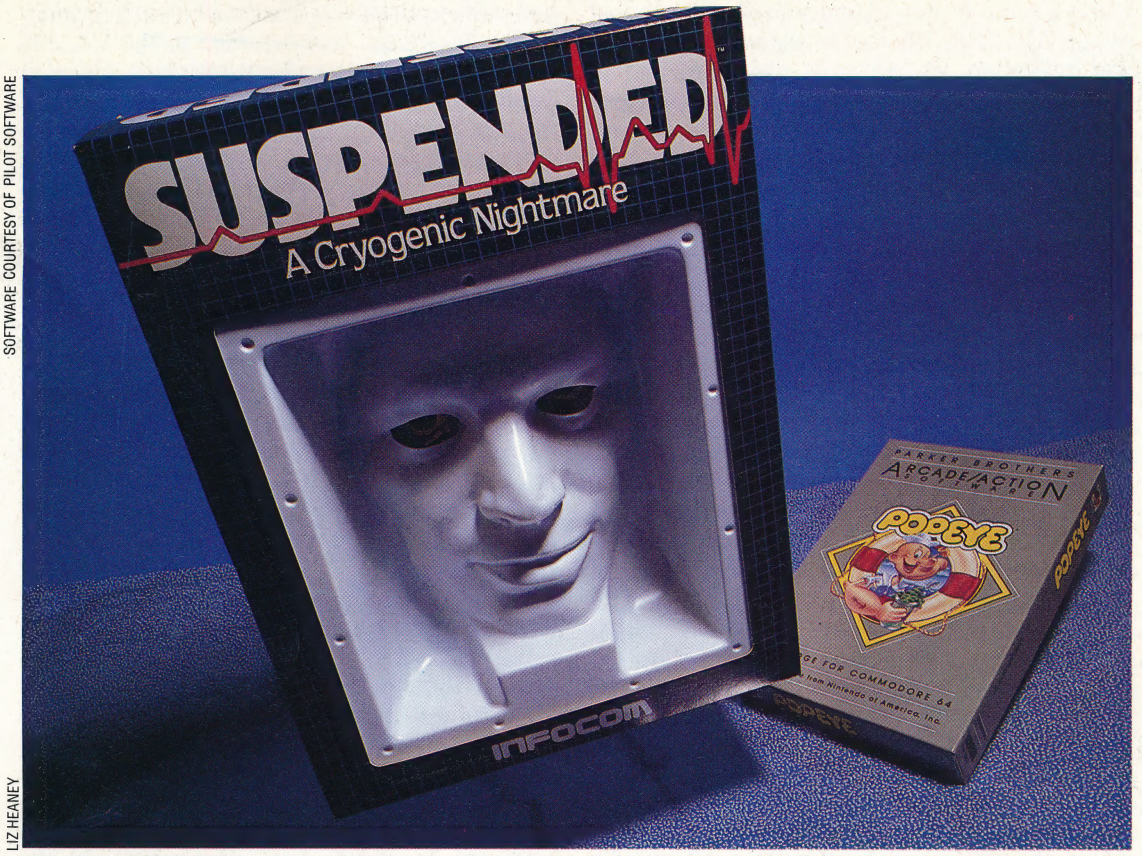
How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

SELLING THE SIZZLE



SOFTWARE COURTESY OF PILOT SOFTWARE

LIZ HEANEY

You Can't Judge A Book...
These two disk-based programs cost about £40 each, but one has been packaged in an imaginative way that attracts the eye and suggests value for money, while the other, by comparison, looks like a slightly dull box at a very high price

Now that home computers are big business, it is increasingly marketing, rather than manufacturing, that makes or breaks a product. Whether you are selling a games package or a new business computer with a considerable marketing budget, the crucial factor is creating the right 'product image'.

Marketing is the business of building a bridge between the product that the sellers have to sell and the pockets of the would-be buyers. In designing a bridge that will encourage and support the maximum amount of traffic, the marketeers have to make guesses (sometimes based on market research) about the needs, desires and whims of their public, and then back those guesses with heavy investment. For example, the recent Macintosh launch at its peak was costing £1 million a month in the UK alone.

Marketing decisions begin to be made when the production of a new machine is in the planning stage. What functions the machine will have, how many units are to be built, and how much can be spent manufacturing each unit are all factors that will affect the marketing.

Similar considerations apply to software marketing. It is no good spending huge sums developing and selling a game that will have to cost more than its potential buyers can afford if it is to recoup its investment. But a software house that stints on development will be left having to sell a product that is short on features, or bug-ridden, or both, thus running the risk of bringing its brand name into disrepute, with a potentially harmful effect on future products. If it spends a lot on development but nothing on marketing, it will be left with a splendid product that nobody knows about.

It is also important to determine at an early stage where the product fits in relation to similar goods available in the shops. It is here that the 'image' of the product becomes vital. The makers of cigarettes and soap powder share a difficulty — if the truth be told, one product is pretty much like another. Computer hardware manufacturers are not quite in this position, but it isn't always easy to explain precisely the individual character of a machine to first-time buyers who will probably not understand the fine technical points. Games and business application software, too, often reveal their best features only after they've been bought and used.

For this reason, hardware and software manufacturers, like the makers of cigarettes and soap powder, resort to the creation of a 'product image', a sort of short-hand psychological projection of the product. It is more effective to

sell an idea than a mere product, a principle embodied in the venerable advertising slogan that you 'sell the sizzle, not the sausage'.

The 'industrial design' of the machine (its outward appearance, and the external layout of its switches and function keys) is often a useful starting point in developing a hardware product image. The BBC Micro, for example, is unfussy and utilitarian, as befits its claims to special educational value. The casing of other machines may be tooled with 'go-faster' mouldings and jazzy logos to entice the games player. The Atari XL range — restyled as part of a marketing effort to rescue the product from the doldrums in mid-1983 — has a rugged, squared-off styling edged with castellated air vents that emphasise its military appearance — just right for playing Tank Battle. Commodore, in their own battle for the US market, commandeered Ferdinand Porsche, designer of the famous car body, to enhance their range with an elegantly rounded design intended to be confidently, but not obtrusively, futuristic.

In contrast, the physical appearance of the Sinclair Spectrum, with its small black casing and elaborately inscribed multi-function keys, implies a machine that crams a lot into a little, with a strong suggestion of value for money.

Product image goes further than considerations about the machine's physical appearance. The image must be boosted by a coherent advertising 'platform' that reinforces the idea. The supposed 'jumbo memory' of the Commodore 64 is put across by introducing an elephant into television and magazine advertising. For the BBC, the logo of the owl evokes the idea of 'wisdom' in the public mind.

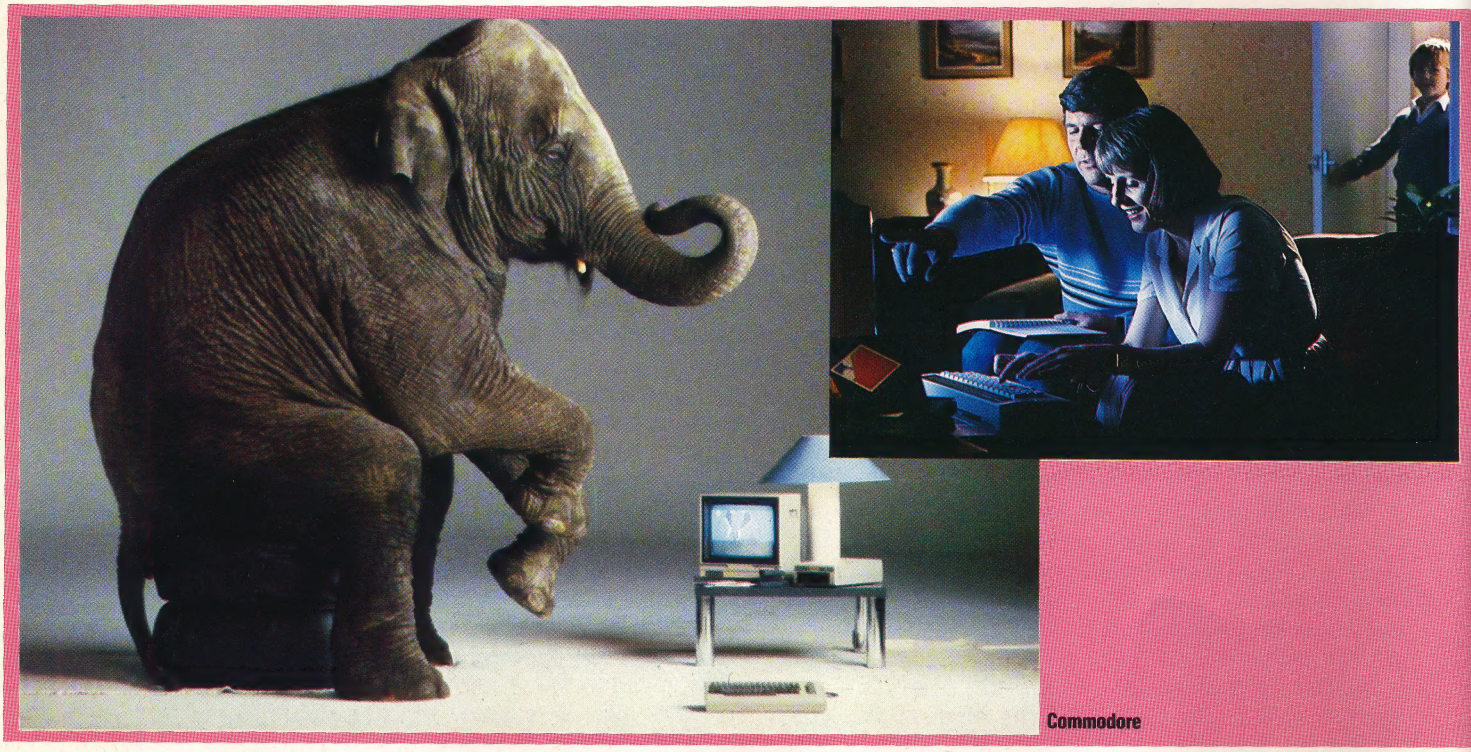
Naturally, the name is an important consideration. Early home computers had to fight hard against an image entrenched in the public mind by films of the 60s and early 70s, where the computer was invariably depicted as an inhuman 'Big Brother' beyond the control of mere individuals. Accordingly, micros were given domestic names that deliberately defused their high-tech associations. Hence PET and Apple.

The Macintosh was heralded in the UK and the US by a 'teaser' television campaign, filmed in England by Ridley Scott, that showed a 'Big Brother' screen being smashed by a woman to represent the new-found freedom offered by the latest Apple product. The slogan was 'Thanks to the Macintosh, 1984 won't be like 1984'. To the American market 'Mackintosh' is a well-known variety of apple. The development team, for whom spelling was not a strong point, dropped the 'k' and the mistake stuck.

But people need to be assured that they are not just being sold a toy. The PET ran into a good deal of sales resistance on this point as Commodore tried to develop into the business market in the late 1970s. The first line of attack was to suggest that the name was an acronym for the much more scientific-sounding Personal Electronic Transactor, but the ploy wasn't very persuasive, and they reverted to renaming the micro the Commodore Business Computer.

The high-tech tradition enjoyed a vogue as the public became increasingly comfortable with the idea of having computers around. The effectiveness of product names in this category depends on moving them as far from everyday language as possible, so that acronyms and even collections of letters with no meaning are preferred to words you might find in the dictionary. The rarer letters that score high in Scrabble also score high here. Hence machines

Images
The importance of creating a powerful product image in marketing is clearly seen in our example: Commodore's elephant suggests the 'jumbo' memory of the Commodore 64, the BBC's home and school scenes convey the Electron's flexibility, friendliness and educational importance, while the rather more fanciful Apple scenario with its Orwellian allusions promises Apple users freedom from drudgery in a humanised computer world. Like most high-power techniques, however, they can have unforeseen effects: the elephant is proverbially terrified of mice (unlike the Macintosh and Lisa), and might suggest a product that is old fashioned; potential Electron buyers may find the family setting cloying and stereotyped, and the school connection intimidating; Apple customers might think that they were being portrayed as mindless clones



Commodore

like the ZX81, the MTX500 and the MZ-700.

Packaging, merely protective in the case of hardware, has a very special importance for software's image. Broadly, two strategies are available to software vendors: to spend only the minimum on packaging (the cassette box with a coloured leaflet enclosed) and a corresponding 'bargain' price, or to build up the product's 'perceived value' by wrapping it in an enlarged case, often made to look like a book, and adding extra goodies that may or may not be part of the game. The Hobbit, for example, comes with the paperback edition of Tolkien's famous book.

The image is projected through advertising. Broadly, the relationship between marketing and advertising is the relationship between strategy and tactics. The questions of when to advertise, and how big the advertising budget should be, are marketing decisions. Typically, a home computer manufacturer will launch a new product shortly before the Christmas season and put a large proportion of the year's advertising budget into supporting the launch.

But promotion of the product image isn't always helpful to sales when the image is wrong. One marketing campaign, for cigarettes, is now legendary in the advertising business. The brand name 'Strand' was widely promoted in cinema and television advertisements associated with a solitary man in a white raincoat, and the copy platform was 'You're never alone with a Strand'. But the message that came across was 'Loners smoke Strands', and the brand was shunned.

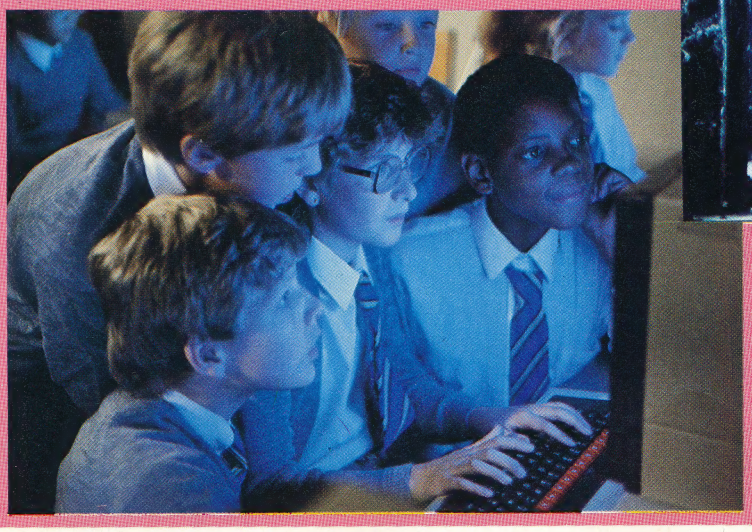
Some current computer images may be backfiring in the same way. The Commodore elephant may serve as a reminder that the 64 is unwieldy for writing programs. Many distributors work hard to play down the BBC's strong educational associations, fearing a dry-as-dust flavour may drive away trade.

You might call the establishing and projection of images the 'creative' side of marketing (advertising people do). But the logistics of marketing is every bit as important — more so, perhaps, as this is so frequently the weak link in the chain. It is one thing to stir up the public imagination about a new product; to get it into their homes or offices is a more difficult proposition.

Is it to be distributed by mail order, in the manner of Sinclair? This is a cheap way of getting goods to customers, but how do you assure them of your support? You will probably have to sell at bargain prices to counteract the reassurance of buying in a shop, but if you cut your margins too thin you may have difficulty in financing volume production, with corresponding delays in delivery.

Selling through the established chains such as WH Smith, Dixon's or Curry's will give customers a strong sense of security about their purchase, but these chains will exact a penalty in big mark-ups, again cutting back profits. A software package that retails at £4.99 may bring you less than £2.25 once the chain has taken its cut. Alternatively, you might think of setting up a network of hand-picked dealers, each one able to give your customers special help and advice with your product.

In the struggle for survival in the micro world, the demise of some manufacturers of excellent hardware and software products has brought home the fact that making goods is often less than half the battle. Making and maintaining markets is where the story really starts. And where the buck really stops.



Acorn/BBC

Apple

FLOWER POWER

Daisy wheel printers produce print of a far higher quality than their dot matrix rivals, and allow useful features such as proportional letter-spacing. For a home computer owner, however, the need for quality print-outs may be overshadowed by the comparative expense and slow speeds of these printers.

At first sight, a daisy wheel printer might seem a strange buy for a home computer owner. It is not really suitable for listing programs, it is slow, and it costs more than a dot matrix printer. Nevertheless, for some applications it is a good choice. The area in which a daisy wheel printer wins hands down is in the quality of the print. A dot matrix printer builds up each character by printing a pattern of dots: no matter how many pins are used in the print head, the individual dots can still be seen in the printed text.

With a daisy wheel, on the other hand, the characters are produced by a type block hitting an inked ribbon — just like a typewriter. These type blocks, one for each character, are arranged in a circle rather like the petals in a flower — hence the name *daisy wheel*. The resulting print is easier to read and also looks more 'professional'.

The penalty to be paid for this higher quality print is in the printer's speed: daisy wheel printers are much slower than comparably-priced dot matrix printers. The reason for this lies in their different printing methods. To print a character using a daisy wheel printer, the print wheel is first spun until the required 'petal' is at the top, then the print hammer hits the type block, and the carriage is moved on to produce the next character.

Compare this with a dot matrix printer, where the dots are printed as the carriage moves across the paper, and the difference in speed between the two types of printer is understandable. A £400 daisy wheel prints about 20 characters per second (cps); an Epson FX-80 dot matrix printer costs about the same, yet can print at a speed of 160 cps. Some daisy wheel printers are faster, but they cost more — over £1,000 for an 80 cps model.

To increase the print speed, both daisy wheel and dot matrix printers often have two extra features: bi-directional printing and logic seeking. *Bi-directional* simply means that one line of text is printed from left to right, and the next line is printed from right to left. The printer does not have to wait for the carriage to return and it can therefore print faster. *Logic seeking* means that the carriage skips spaces to reach the next word in the text — less sophisticated printers take the same time to 'print' a space as any other character.

Dot matrix printers have their character shapes stored in ROM memory inside the printer; daisy wheel printers have the character stored as type blocks on the print wheel. Each method has its advantages: with a dot matrix printer, the character shapes can be re-defined by sending suitable escape codes to the printer from your micro. With a daisy wheel printer the process is much easier: simply swap the current daisy wheel for a different one.

Daisy wheels come in a variety of type styles and pitches; the *type style* refers to the design of the characters, and the *pitch* refers to their width. Some of the more common type styles are Courier, Roman, Gothic and Italic. The pitch is normally 10 or 12 characters per inch. A plastic daisy wheel costs about £5, whereas a metal version costs over £20; metal wheels have the advantage over plastic of lasting much longer.

The one problem with all these different type styles is that few of them are exactly the same as the character set used by a micro. This means that some of the characters on your micro keyboard will be printed out as something completely different. Often the 'hash' character ('#') prints as a pound sign ('£'), or a square bracket ('[') prints as a fraction ('½'). With many type styles the number '0' (zero) is indistinguishable from the capital letter 'O' and similarly the lower case letter 'l' may be mistaken for the number '1'. While the more expensive daisy wheel printers have 127-character print wheels, most can only print 92 or 96 characters, and it's these that suffer most from this type of problem.

As you can imagine, trying to debug a program is not made any easier if you are not sure which characters are 1's and which are l's. For this reason, and also because of its slowness, a daisy wheel printer is not recommended if you use your computer mainly for programming.

SPECIAL EFFECTS

A daisy wheel printer can be programmed to produce a variety of special effects, just like a dot matrix printer. Although the number of these effects is limited, the method of programming the printer is identical to that used for a dot matrix printer, namely by sending escape codes (see page 324). For example, on a Diablo daisy wheel printer the ESC-E code turns on the automatic underlining, and ESC-R turns it off again. Using standard Microsoft BASIC, you would type LPRINT CHR$(27);"E"; and LPRINT CHR$(27);"R"; to send the above codes to the printer. Other codes include ESC-1 to set a tab stop; ESC-9 to set the left margin;

and ESC-U to feed the paper up half a line — useful for subscripts. The daisy wheel equivalent of emphasised text on a dot matrix is called *emboldening* — where each character is printed four times to make it stand out. The Diablo code for this feature is ESC-O. Emboldening script, however, drastically reduces the print speed.

Some daisy wheel printers allow you to vary both the distance moved between characters and the line spacing. If this is so, then the printer can be used to create graphic images or screen dumps, just like a dot matrix printer (see page 344). If a screen pixel is 'on' then the daisy wheel prints a full stop, if it's 'off' then a space is printed. By reducing the distance moved between characters, a whole horizontal line of the screen can fit across the paper. Similarly, the line spacing is reduced so as not to leave a gap between the lines. The process is, however, very slow.

One feature not seen on dot matrix printers is the ability to centre headings automatically. Sending ESC = to a Diablo printer will centre the rest of that line between the margins. Another novel feature is the decimal tab: ESC-H will cause all numbers to be printed with the decimal points aligned, which is a useful feature for sums of money.

The more expensive daisy wheel printers can usually perform proportional spacing. With this feature the pitch is not a standard 10 or 12 characters per inch. Instead, it varies depending on the width of the character being printed. For example, the character 'w' is much wider than the character 'i', so with proportional spacing the carriage would not advance as far for an 'i' as it would for a 'w'. What this means is that characters in successive lines of text do not fall exactly underneath each other, and the overall effect is more pleasing to the eye. The lines of text that you are now reading are proportionally spaced. Often a special daisy wheel is necessary to use this feature correctly, but it is turned on and off using escape codes — like any other printer effect. On a Silver-Reed EXP 770 printer, the codes ESC-P and ESC-Q turn the proportional spacing on and off.

For a business, a daisy wheel printer can easily be justified; for most home computer owners it probably can not. There is an alternative though, and that is to adapt an electronic typewriter. Daisy wheel printers cost more than electronic typewriters, even though they both use the same print method. Until recently, however, typewriters could not easily be connected to a computer — they had no interface circuitry or RS232 socket. But now every electronic typewriter on the market either comes with a built-in computer interface, or can be fitted with an interface kit for about £100. The great advantage in adapting an electronic typewriter, apart from saving about £200 over a comparable daisy wheel printer, is that it can still be used as a conventional typewriter. So you have both a typewriter, for typing a short letter or addressing envelopes, and a daisy wheel printer for word processing.

Quality At A Price

Print sample with 10 characters per inch
Print sample with 12 characters per inch
Print sample with 15 characters per inch

A half-line feed is used for subscripts:-
H_2O

ESC-E turns on the automatic underlining and ESC-R turns it off.

Bold print makes text **stand out** from the rest of the text.

The ESC= code is used to centre text:-

A Centred Heading

These lines of text were printed without the proportional spacing feature. Note especially how the numbers 0123456789 are spaced out. Without proportional spacing, the width of each character is the same. For example, a 'W' is the same width as an 'i':-
WWWWWWWWWWW
iiiiiiiiiii

These lines of text were printed using the proportional spacing feature. Note especially how the numbers 0123456789 are spaced out. Using proportional spacing, the width of each character varies. For example, a 'W' is much wider than an 'i':-
WWWWWWWWWWW
iiiiiiiiiii
The overall effect is to make the text more attractive.

More expensive and less flexible than a dot matrix printer, the daisy wheel produces typewriter-quality printing and proportional spacing. The wheel itself (distinctly flower-like) is easily replaced by another in a different typeface or style

READ ALL ABOUT IT

When Sinclair Research designed Spectrum BASIC, the company took care to allow more experienced programmers to bypass some of its limitations. Here we present an explanation of how to read directly from the Spectrum's keyboard, and give a simple program to move a 'graphics pen' under keyboard control.

BASIC allows you to enter information from the keyboard using INPUT and INKEY$. The INPUT statement reads a number or string of characters, terminated by ENTER, to a numeric or string variable. The INKEY$ function scans the keyboard and returns either a string containing the character of the key pressed or, if no key is pressed, an empty string. These are both useful commands, but for some applications — for example, when combinations of keys must be read simultaneously — the IN function can be used to read the keyboard directly.

The keys on the Spectrum are connected to the Z80 microprocessor through input/output ports. There are 65536 input/output ports, and each one can be addressed individually. In the same way that PEEK and POKE are used to read or write to memory, IN and OUT are used to read and write to input/output ports. The function IN(m) will return the value of the port m, while the statement OUT(m,n) will write the value n to port m.

The keyboard consists of four rows of 10 keys, each row being divided into two half-rows of five keys. Each half-row maps onto a port as shown. Notice that the keys in the left-hand half-rows map to their ports from right to left, but that the right-hand half-rows map left to right.

The rightmost five bits of a port each map onto a key, and will take the value 0 if the corresponding key is pressed, or 1 if it is not. The bits marked X are unspecified — they can hold either a one or a zero. This means that the value of IN(m) will be undefined. This problem can be overcome by setting the value of the top three bits to zero in software by using the following instruction:

```
DEF FN a(p) = p - INT(p/32)*32
```

where p is the value in the port — the undefined byte returned by IN(m). The division p/32 will give a number from 0 to just under 8. Taking INT(p/32) will discard the fractional part of this number, leaving an integer value from 0 to 7. Multiplying this by 32 then gives a value representing the top (leftmost) three bits of p.

In Touch
The Spectrum keyboard comprises eight blocks of five keys, each block being constantly monitored by a dedicated byte, or 'port'. Each key maps onto one bit of its port; while a key is held down, its 'signal' bit switches from one (its normal state) to zero. Here, keys Y, I and O are being pressed so the binary value of their port, byte 57342, is XXX01001 — bits 5 to 7 are undefined. Similarly, A and S are being pressed, so the value of byte 65022 is XXX11100

KEVIN JONES

Subtracting this from p itself will remove these three bits, leaving an integer value between 0 and 31, representing the bottom (rightmost) five bits of p. The value of FN a(p) will therefore always be defined with the top three bits set to zero, whatever the value of the top three bits of p.

For example, if the letter V is pressed, port 65278 will contain:

X	X	X	0	1	1	1	1

FN a(IN(65278)) will therefore return the value:

0 0 0 0 1 1 1 1 = 15

If both Caps Shift and the letter V are pressed simultaneously, the port will contain:

X	X	X	0	1	1	1	0

and FN a(IN(65278)) will therefore return the value 14. If no keys are pressed at all, the port will return the value 31 (00011111).

Try this program:

```
10 REM Print out the masked value of the ports
20 REM Define the masking function
30 DEF FN a(p)=p - INT(p/32)*32
40 REM Print out the ports
50 PRINT AT 1,1;"Port              Masked value"
60 PRINT "32766",FN a(IN(32766))
70 PRINT "49150",FN a(IN(49150))
80 PRINT "57342",FN a(IN(57342))
90 PRINT "61438",FN a(IN(61438))
100 PRINT "63486",FN a(IN(63486))
110 PRINT "64510",FN a(IN(64510))
120 PRINT "65022",FN a(IN(65022))
130 PRINT "65278",FN a(IN(65278))
140 FOR i=1 TO 250: NEXT i
150 CLS
160 GO TO 50
```

After entering RUN, try pressing some keys and notice how the masked values of the ports change. Try to predict the values that will be displayed for different keys and combinations of keys. If you press the cursor keys (5, 6, 7 and 8) you should get the following masked values on ports 61438 and 63486 respectively (which we will call port A and port B

	Port A (port 61438)	Port B (port 63486)
←	31	15
↓	15	15
↑	23	15
→	27	15

If you press the Caps Shift key, you should get the masked value of 30 for port 65278 (which we will refer to as port C).

Consider the simple graphics program given here. This program will read the cursor keys to draw horizontal, vertical and diagonal lines and, if the Caps Shift key is pressed, move a 'graphics pen' without drawing a line. Diagonal lines are drawn by pressing two cursor keys at once.

In the program, line 30 defines a function (FN(a)) to mask off the top three bits, as before. The horizontal position of the graphics pen is represented by x, and y is its vertical co-ordinate. Line 50 sets the pen's initial position at the centre of the screen.

Lines 70 and 80 read in the ports mapped to the cursor keys. The half-row from 6 to 0 includes three of the cursor keys and maps to port 61438 (port A). The half-row 5 to 1 contains the right arrow cursor key and maps to port 63486 (port B). Line 90 reads the Caps Shift key — the half row from V to Caps Shift is mapped to port 65278 (port C).

Lines 110 to 180 test for the eight 'legal' combinations of cursor keys, and the x,y position of the graphics pen is adjusted accordingly:

Line 110 tests for ↑
Line 120 tests for ↑ and →
Line 130 tests for →
Line 140 tests for ↓ and →
Line 150 tests for ↓
Line 160 tests for ↓ and ←
Line 170 tests for ←
Line 180 tests for ↑ and ←

Lines 200 to 240 ensure that the pen does not move off the edge of the screen. Finally, if the Caps Shift has not been pressed, line 250 will plot the graphics pen's new position on the screen. Line 260 then loops back to repeat the whole process again.

Shifty Business
The program employs the techniques discussed for detecting multiple keypresses: the unshifted arrow keys allow you to draw horizontal, vertical and diagonal lines (diagonal= horizontal+vertical), while the shifted keys produce the same cursor movements without screen plotting. The same techniques will be used to better effect in a future article

Detecting Multiple Keypresses

```
 20 REM Set Up Function to Mask Top Three Bits
 30 DEF FN a(p)=p-INT (p/32)*32
 40 REM Initialise Pen Position
 50 LET x=127: LET y=35
 60 REM Read Ports and Mask Off Top Three Bits
 70 LET portA=FN a(IN 61438)
 80 LET portB=FN a(IN 63436)
 90 LET portC=FN a(IN 65278)
100 REM Alter Pen Position Depending on Which Key
       Has Been Pressed
110 IF portA=23 AND portB=31 THEN LET y=y+1
120 IF portA=19 AND portB=31 THEN LET x=x+1:
    LET y=y+1
130 IF portA=27 AND portB=31 THEN LET x=x+1
140 IF portA=11 AND portB=31 THEN LET x=x+1:
    LET y=y-1
150 IF portA=15 AND portB=31 THEN LET y=y-1
160 IF portA=15 AND portB=15 THEN LET x=x-1:
    LET y=y-1
170 IF portA=31 AND portB=15 THEN LET x=x-1
180 IF portA=23 AND portB=15 THEN LET y=y+1:
    LET x=x-1
190 REM Stop Pen Going Off The Screen
200 IF x<0 THEN LET x=0
210 IF x>255 THEN LET x=255
220 IF y<0 THEN LET y=0
230 IF y>175 THEN LET y=175
240 REM Plot Point if CAPS SHIFT Not Pressed
250 IF portC=31 THEN PLOT x,y
260 GO TO 70
```

CRASH

When a microcomputer *crashes*, it goes into a state equivalent to a coma — the machine is still 'alive', the screen may even continue to display text or graphics, but normal functions cease and keyboard input is not recognised. In this situation, the user must either press the Reset button (if one is fitted) or disconnect the power supply — which usually results in the loss of all programs or data held in RAM. A crash is therefore often referred to as a *fatal error*.

Microcomputer crashes have three main causes. The power supply is often the culprit; the earliest micros in particular were extremely susceptible to small fluctuations in mains voltage. Such fluctuations could be caused by inconsistent supplies from the local generating station, or by simply plugging in another appliance close to the computer's mains socket. Recent microcomputer designs incorporate more efficient 'smoothing' circuits into the power transformer, but their effectiveness depends on the local mains supply — if your area is prone to mains 'spikes' (bursts of higher voltage) or 'brown-outs' (the lights flicker and become dimmer for a few seconds), then it might be a good idea to invest in an external smoothing unit.

Another cause of a crash is an infinite loop within a program. In BASIC, this is hardly a problem — `100 GOTO 100` is an infinite loop, but it is quite simple to break out of by pressing the Break or Stop key. Few machines are able to interrupt a machine code program from the keyboard, so a JMP loop with no exit condition will cause the system to crash.

Most microprocessors will crash if they are confronted with an op-code that does not correspond to anything in their instruction set. This may result from a program that has been wrongly assembled — perhaps a JMP instruction has been misused so that the processor mistakes a piece of data for program code.

CROSS-ASSEMBLER

A *cross-assembler* is a utility program that is used to assemble source code into object code that will run on a different type of machine. For example, a cross-assembler may be used on a BBC Micro to develop machine code programs for a Commodore 64. This should not be confused with converting a BBC program to run on the 64 — a far more difficult exercise.

Cross-assembly allows a programmer to develop code for many different machines by using one system. This development system will probably have a better keyboard and screen and more RAM than a home computer. It will almost certainly execute code faster, and will have a library of editing and utility programs that are not available on the target machines.

The development system may even have a different processor from the one on the target machine — the cross-assembler can deal with the different instruction sets and will ensure that the assembled code is located in the correct memory area. Many home computer programs are in fact developed on mainframe computers. A *cross-compiler* works in a similar way, but is used to compile high-level languages rather than Assembly code.

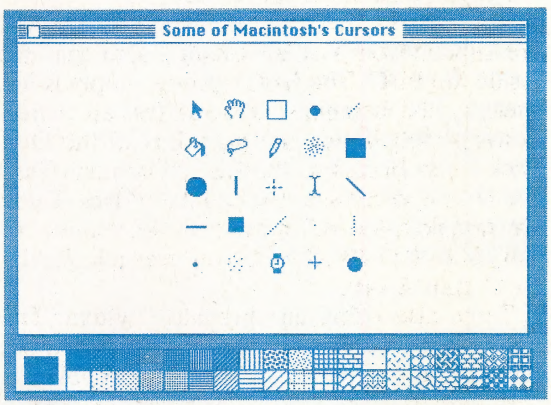
CURRENT LOOP

The most widely established standard for serial data communication is RS232, which specifies the format for the data bits (including start and stop bits), plus the 'handshaking' lines to control the transmission. ('Handshaking' is the name given to the procedure in which, when data is transferred across an interface, one side introduces itself to — or 'shakes hands' with — the other in a pre-defined sequence of actions.)

Within the RS232 standard, there are two variations: V24 and 20 mA (milli-amp) *current loop*. The first of these is the most common; here the data bits are represented by voltage levels, with the difference between a zero and a one bit being 24 volts. Current loop was used in the original teletype terminals. When this system is used, the data line and the ground (or earth) line form a loop between two peripherals, and a current of 20 mA signifies a one while an absence of current denotes a zero.

CURSOR

A computer cursor is simply an on-screen symbol that indicates where the next character will appear. Its name was originally derived from the Latin (it means 'to run before' or 'to herald'), but it was more recently used to describe the movable marker on a slide rule or the pointer on a mechanical adding machine.



Microcomputer cursor symbols have traditionally been either a flashing square or an underline character. Some have simply been a flashing letter of the alphabet (the Sinclair Spectrum displays 'K', 'C', 'L', or 'G' depending on the current mode), but on machines like the Apple Lisa and Macintosh the nature of the cursor changes according to the task being performed. For example, a wristwatch symbol indicates that the user must wait, and a small picture of a hand is used to point to an object that can be moved.

LEADING LIGHT

The BBC Micro is a computer that is severely limited by its memory size. For business use, the computer must be augmented. The Torch Disk Pack does this by adding an extra 64 Kbytes of RAM memory, two double-sided disk drives and a Z80 microprocessor that allows the BBC to run CP/M-style software.

The BBC Micro is capable of expansion to full business specification. Although this capability was built into the machine from the start, the first company to make use of the idea was not the manufacturer, Acorn, but Torch. The Torch Disk Pack allows the BBC user to run the vast range of software that uses the CP/M operating system. Yet at £804 the whole package costs only a little more than a pair of Acorn disk drives.

The CP/M operating system requires a Z80 microprocessor and disk drives. The BBC uses a 6502 as a microprocessor, but the Torch Disk Pack adds a Z80, leaving the BBC's 6502 to handle all input and output functions. The keyboard, screen and disk drives are all controlled by the 6502, while the Z80, with its 64 Kbytes of RAM, runs the CP/M programs and generally 'takes charge' of the system. Data is passed back and forth between the two microprocessors via the BBC's expansion port or 'tube'.

The Z80 processor and its 64 Kbytes of RAM are supplied on a small circuit board that fits inside the BBC. The BBC's power supply is not used by the combined system. Instead a new power supply, inside the main box of the Disk Pack, is connected to the BBC. The main Disk Pack box also contains the two disk drives. These are double-sided 80-track drives that are very similar to the 800 Kbyte drives available for the BBC from Acorn.

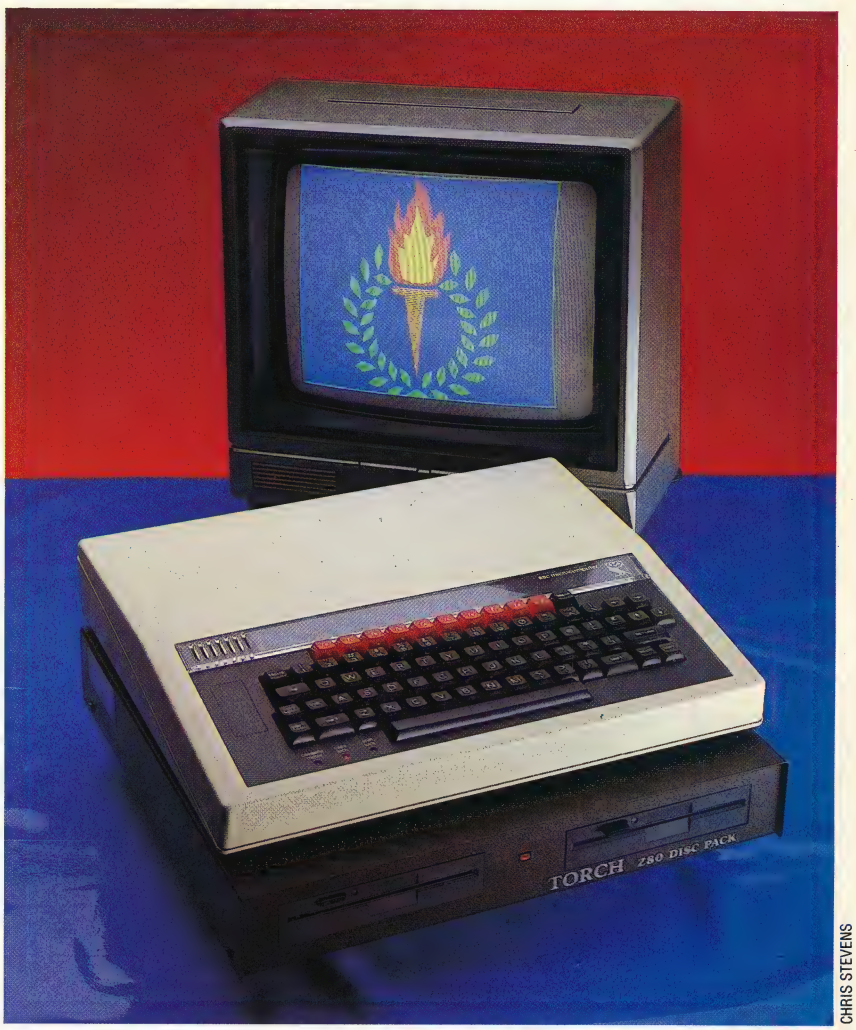
Torch also offers an alternative system. The ZEP 100 is essentially a Torch Disk Pack without the disk drives that is intended for BBC owners who already own disks. Torch also markets a range of business computers that are made up of a BBC Micro board and disk drives, with a built-in monitor and modem.

The Disk Pack is designed to fit underneath the BBC, with short cables connecting the two together. The box is exactly the same size as the computer, so the whole system has a neat appearance. Unfortunately, the Disk Pack raises the BBC keyboard about three inches off the desk, which can make typing on the complete system a little difficult.

The Torch Disk Pack comes supplied with all necessary parts to upgrade the BBC, with one important exception — it does not include the set of disk interface chips needed for the BBC to use a disk drive. A few people will have paid for these when they bought the BBC Micro, but most will have bought a standard BBC Micro Model B and so will have to pay £100 or so to add the interface chips. Model A BBC Micros must be upgraded to Model B standard to use the Torch Disk Pack.

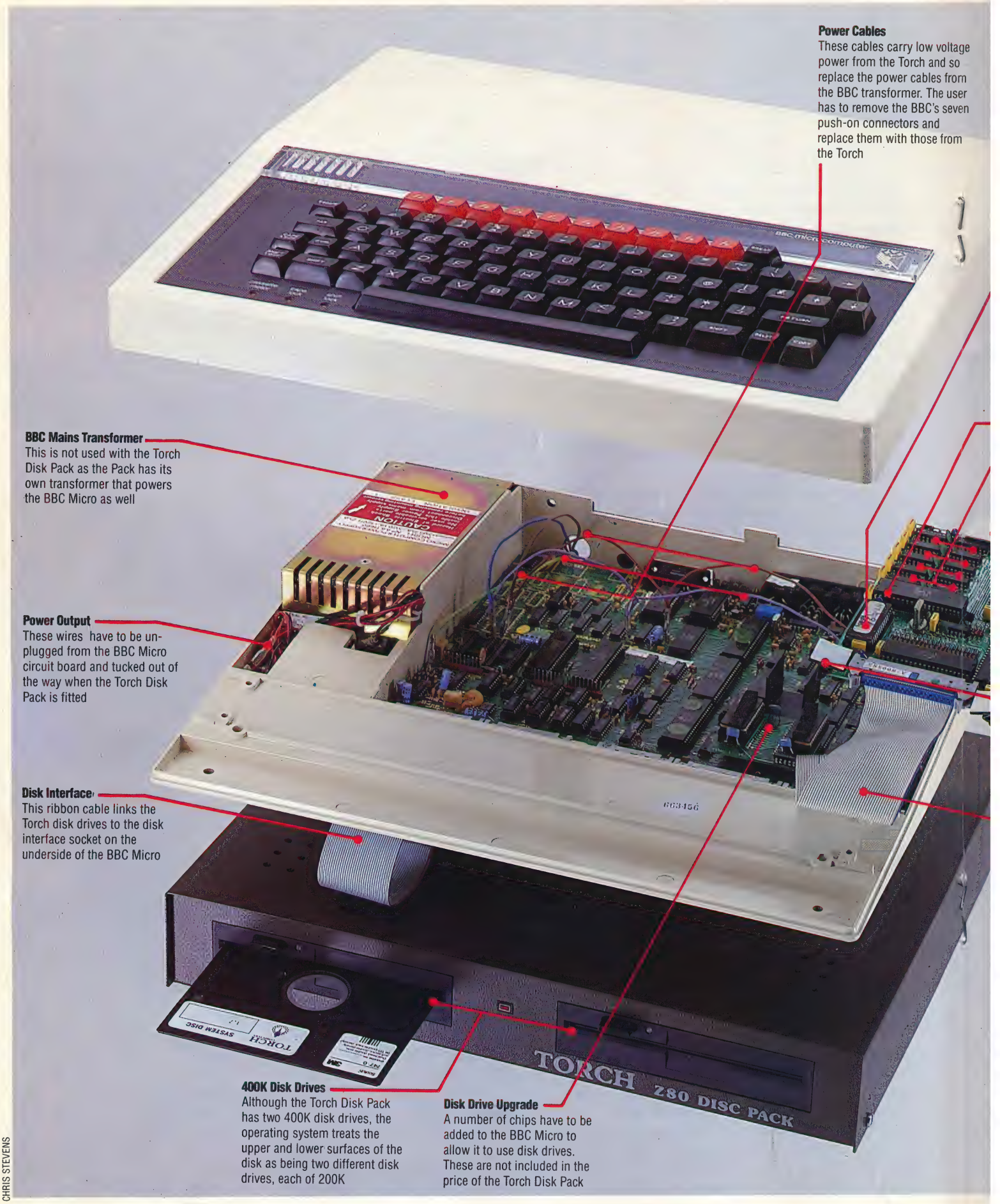
Once the Disk Pack is correctly fitted and the system switched on, it starts up straight away (in display mode 3) as a CP/M computer. The disk operating system used is actually written by Torch and is called MCP. This is very similar to CP/M and will run most of the hundreds of CP/M programs. The main difference between the systems is that MCP is stored in a ROM inside the BBC and does not need to be loaded into the micro.

There are many commands available in MCP that will be familiar to BBC users. Different



CHRIS STEVENS

BBC Upgrade
The Torch Disk Pack turns the BBC Micro into a business computer because it includes a Z80 processor with 64K of memory that allows it to use an operating system similar to CP/M. It includes two 400K disk drives and yet the whole system costs £804, little more than a pair of Acorn disk drives



Power Cables
These cables carry low voltage power from the Torch and so replace the power cables from the BBC transformer. The user has to remove the BBC's seven push-on connectors and replace them with those from the Torch

BBC Mains Transformer
This is not used with the Torch Disk Pack as the Pack has its own transformer that powers the BBC Micro as well

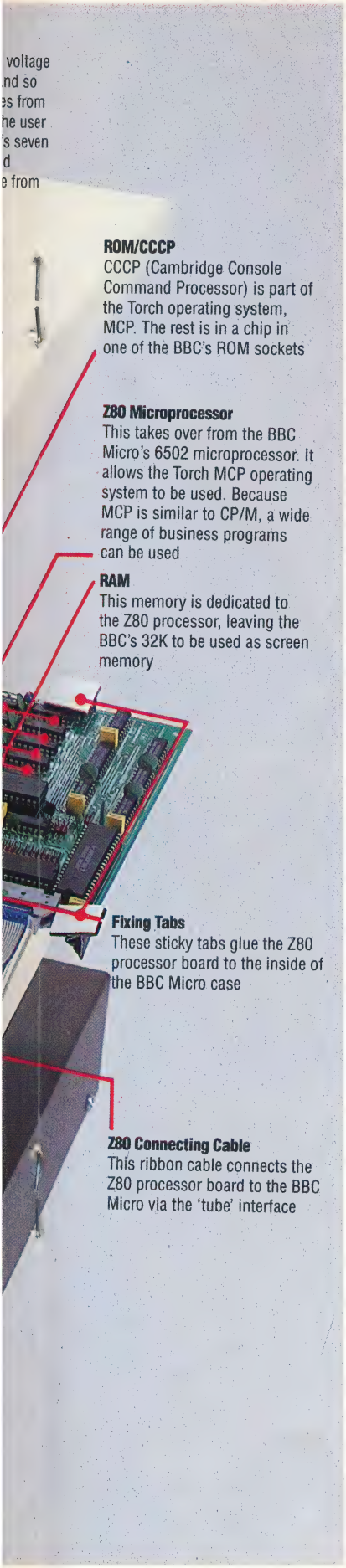
Power Output
These wires have to be unplugged from the BBC Micro circuit board and tucked out of the way when the Torch Disk Pack is fitted

Disk Interface
This ribbon cable links the Torch disk drives to the disk interface socket on the underside of the BBC Micro

400K Disk Drives
Although the Torch Disk Pack has two 400K disk drives, the operating system treats the upper and lower surfaces of the disk as being two different disk drives, each of 200K

Disk Drive Upgrade
A number of chips have to be added to the BBC Micro to allow it to use disk drives. These are not included in the price of the Torch Disk Pack

CHRIS STEVENS



display modes can be selected with MODE and all the VDU and *FX commands are there. *KEY is still used to define the function keys. Four of the function keys are pre-defined with a selection of useful commands.

There are many other commands that deal with loading and manipulating files on disk. Apart from being able to load machine code programs by simply typing their names, the system offers a command to load special files that build up pictures on the disk using the BBC operating system. PRINT outputs a text file to a printer, while TYPE outputs it to the screen. COMMAND uses a file as a series of commands for the computer in a similar fashion to EXEC in BBC BASIC.

There are several other utility programs that are kept on a system disk that accompanies the Disk Pack and are loaded into the machine by typing the relevant name. These include a routine to change the font (design) of the characters on the screen, a music writing utility, a machine code debugger, a disk editor, and a utility to allow the Torch to read Acorn disks and vice versa. The latter is very useful as the formats used for the two systems are different. This utility allows you to, say, produce a text file using a BBC BASIC program to input text and then to edit it with a CP/M word processing program.

Although the Torch disk format is different from the Acorn format, it uses the same strange convention of treating the two disk drives as being four different disk drives.

Despite all these enhancements, the system may still be used as a standard BBC Micro. Typing *BASIC makes the system ignore all the extras supplied by the Disk Pack. Strangely, the BBC Micro that you are left with is set up to use cassettes and not disks. Despite the fact that it is now fully equipped to use the disk drives of the Disk Pack, without using MCP, the user must specify that the disk filing system is required. When in BBC BASIC you can change back to using the Z80 at any time by turning the machine off and on, or by typing *MCP.

The BBC Micro is a home computer with many facilities, but it *is* only a home computer. For business use a lot more is required of a micro than the BBC alone can offer. When a Torch Disk Pack is added to your BBC Micro, you still have the home computer there, ready for use at a moment's notice, but, in addition, your computer has all the extras needed to turn it into a real business machine.

Fitting The Disk Pack
Fitting a Torch Disk Pack to a BBC Micro is quite a complicated task. Your computer must be a BBC Micro Model B, fitted with a disk drive interface. As the BBC's power supply is no longer required, all seven push-on connectors from this to the BBC's main circuit board must be removed. These are then replaced by a cable from the power supply in the disk pack.

A ribbon cable from the Disk Pack fits into the BBC's disk drive interface. The ROM containing the MCP operating system is then plugged into a spare ROM socket in the BBC

CHRIS STEVENS

Free Software
The Torch Disk Pack comes with several software packages, supplied at no extra cost. A suite of business software is provided – Perfect Writer (a word processing program), Perfect Speller (a spelling checker program), Perfect Filer (a database program) and Perfect Calc (a spreadsheet program).

In addition to these packages, a version of BBC BASIC is provided that will run on the second Z80 processor. All commands are the same as the version in ROM in the BBC with the exception of the built-in 6502 assembler of the BBC. As the second processor is a Z80 and not a 6502, there is a Z80 assembler included in Z80 BBC BASIC. The bonus is 48K of free memory, no matter which display mode is being used. The BBC Micro has a mere 9K free in some display modes and never has more than 28K free

DFS Chip
A number of chips must be added to the BBC's circuit board to enable it to use a disk drive. Most dealers will fit them for around £100. These chips are not included with the Torch Disk Pack and so must be bought separately. Such upgrades are not intended specifically for the Torch and they contain a chip called the DFS that is not actually used by the Torch. However, it's well worth having, because the DFS (disk filing system) is the operating system used by Acorn. This means the Torch Disk Pack can be used as a pair of disk drives to run BBC Micro disk software on the 6502 processor.

The Torch and Acorn disk formats are not compatible so MCP programs cannot read Acorn disks and vice versa. A program is supplied with the Torch that translates between the two formats

Business Options
The first computer Torch produced was a business machine costing £3,395. Torch also produces versions of the business machine that uses the Unix operating system. This is known as the 700-Series (shown in the photograph). There is also a 300-Series of terminals that link Torch computers in a network

COLOUR BY NUMBERS

This instalment of our series of entertaining program ideas looks at a computer game based on the old party and schoolroom favourite 'Simon Says'. Like its electronic toy counterparts, the program uses colour and sound to provide a challenging memory test.

'Simon Says' is one of the first games many of us can remember. The leader gives instructions such as 'Simon says put your hands on your head' or 'Simon says stand up', and so on. Players are disqualified if they respond to an instruction that doesn't start with 'Simon says...'

Surprisingly, this party or classroom game was turned into an electronic game when a number of dedicated electronic toys, using microprocessors to control a set of buttons, lights and buzzers, were developed. In the electronic version, the computer is the leader and every instruction must be followed.

The toy provides a sequence of tones and lights and the player or players must repeat this by pressing the appropriate keys. The toy then adds another note onto the sequence and starts again. These toys became so popular that people often think of 'Simon' games as purely electronic toys, forgetting their simple origins.

Of course, if you own a home computer, the idea of a dedicated computer toy must seem quite strange. A toy may be very portable, durable and easy to use. But even the best games soon become boring and the computer's ability to run hundreds of different programs ensures that it will never be uninteresting!

The program listed here is called 'Follow That!' and plays a game based on four coloured lights, each with its own sound and key on the keyboard (they are numbered from 1 to 4). The program illuminates one light and then asks you to match it. If you get it right, the program displays two lights, and so on.

There are two ways you can be 'caught out' at this game — you may take too long to respond with the next button or you may hit the wrong button three times in any one game. To make the game more difficult, this version also becomes faster as it goes along, although there is no need for you to repeat the sequence at the speed it was played. The game has a maximum of 50 lights in any one sequence — you are unlikely to reach this limit; most players manage around 15 lights before it all becomes too much for them!

It's worth looking at the way the program is structured. All the colour and sound commands are grouped into subroutines at the end of the program as follows:

```
1000 Display light number a
1500 Get a keypress 1,2,3 or 4
2000 Make a noise for the end of the game
2500 Make a noise to warn that a response was
     wrong
6000 Print a message on line 20 of the screen and
     pause after it if necessary
```

These subroutines might correspond to actual hardware devices on a dedicated toy. Extracting them from the main body of the program has two beneficial effects. Firstly, all the machine-specific complexities of sound and colour are kept in one place, making it easier to move the program from machine to machine. Secondly, it would be easy to use the subroutines for a different game along roughly the same lines. The same program could offer several variations on the game just as a dedicated toy does. You might like to try inventing a few extras of your own and adding these to the program. For example, the program could be adapted to allow individual scores to be displayed for any number of players.

As a computer owner, you should never forget that anything a dedicated toy can do, your computer can do better.

KEVIN JONES

Follow That!

```
  10 REM Follow That! game
  30 LET h=0: LET n=0: LET w=3
  40 DIM c(4): DIM p(4): DIM a(50)
  50 LET p(1)=5: LET p(2)=8: LET p(3)=12: LET
 p(4)=15
  60 LET c(1)=1: LET c(2)=2: LET c(3)=4: LET
c(4)=6
  70 LET s$=" ": FOR i=1 TO 5: LET s$=s$+s$:
NEXT i
  80 LET b$=CHR$ (143): REM  a block
 100 REM *** Instructions page
 110 CLS : PRINT TAB (10);"Follow That!"
 120 PRINT : PRINT
 130 IF n>0 THEN  PRINT : PRINT "You managed
";n;" turns"
 140 IF n>h THEN  PRINT : PRINT "... A new re
cord !!!": LET h=n
 150 IF h>0 THEN   PRINT : PRINT "The best ye
t is ";h;" turns"
 155 PRINT : PRINT "Try to repeat the compute
r's    sequence of lights and sounds"
 160 PRINT "by pressing the keys 1 to 4"
 170 PRINT : PRINT "Press P to play, S to stop"
 180 LET a$=INKEY$: IF a$="" THEN  GO TO 180
 190 IF a$="s" OR a$="S" THEN  CLS : STOP
 200 IF a$<>"p" AND a$<>"P" THEN  GO TO 180
 205 REM  *** New game
 210 CLS : PRINT TAB (10);"Follow That!"
 220 FOR a=1 TO 4: GO SUB 1000: NEXT a
 230 LET n=0: LET m=0: RANDOMIZE
 240 REM *** Next turn
 250 LET n=n+1
 270 LET a(n)=INT (RND*4)+1
 280 IF m=w THEN GO SUB 2000: LET m$="*"+STR $(w)
+" wrong answers!": GO SUB 6000: GO TO 100
 285 LET m$="*Here it comes ...": GO SUB 6000
 290 FOR i=1 TO n
 300 LET a=a(i): GO SUB 1000
 310 FOR j=1 TO 100/n: NEXT j
 320 NEXT i
 330 LET m$="Follow that ...": GO SUB 6000
 340 LET i=1
 350 GO SUB 1500
 360 IF t=0 THEN  GO SUB 2000: LET m$="*Too s
low!": GO SUB 6000: GO TO 100
 370 IF a<>a(i) THEN  LET m=m+1: GO SUB 2500:
 GO TO 280
 380 LET i=i+1: IF i<=n THEN  GO TO 350
 390 IF n>50 THEN  LET m$="*You Win with 50 t
urns!": GO SUB 6000: GO TO 100
 400 LET m$="*Get ready to try again": GO SUB
 6000
 410 GO TO 250
1000 REM *** Light box a
1010 INK c(a)
1015 LET p=(a-1)*8+2
1020 FOR l=10 TO 14
1030 PRINT AT l,p;
1040 IF l=12 THEN  PRINT b$;b$;a;b$;b$;
1050 IF l<>12 THEN  PRINT b$;b$;b$;b$;b$;
1060 NEXT l
1070 BEEP 2/(n+1),p(a)
1080 PRINT AT 10,p;"     "
1090 PRINT AT 11,p;" ";b$;b$;b$;" ";
1100 PRINT AT 12,p;" ";b$;a;b$;" ";
1110 PRINT AT 13,p;" ";b$;b$;b$;" ";
1120 PRINT AT 14,p;"     "
1130 INK 0: RETURN
1500 REM *** Read a key
1510 LET t=250
1520 LET a$=INKEY$: IF a$="" THEN  LET t=t-1:
 IF t>0 THEN  GO TO 1520
1530 IF t=0 THEN  RETURN
1540 IF a$<>"1" AND a$<>"2" AND a$<>"3" AND a
$<>"4" THEN  GO TO 1520
1550 LET a=VAL (a$): GO SUB 1000
1560 RETURN
2000 REM *** Raspberry
2010 BEEP 3,0: RETURN
2500 REM Warning
2510 BEEP 1,0: RETURN
6000 REM *** Print m$
6010 PRINT AT 20,1;s$;AT 20,1;
6030 IF m$(1)="*" THEN  PRINT m$(2 TO ): FOR
z=1 TO 200: NEXT z
6040 IF m$(1)<>"*" THEN  PRINT m$
6050 RETURN
```

Basic Flavours

Commodore 64

Replace CLS by PRINT CHR$(147).
Replace LET A$=INKEY$ by GET A$
Replace RANDOMIZE by XX=RND(-TI).
Replace (RND*4) by (RND(1)*4).
Replace M$(1) by LEFT$(M$,1).
Replace M$(2 TO) by MID$(M$,2).
Replace CHR$(143) by CHR$(166).
Replace PRINT AT L,C; by PRINT LEFT$(DN$,L+1)TAB(C); (e.g. line 6010 becomes
6010 PRINT LEFT$(DN$,21)TAB(1);S$;
LEFT$(DN$,21)TAB(1);
Replace DIM C(4) by DIM C$(4)
Replace b$;b$;a;b$;b$; by BBZBB$; in 1040.
Replace b$;a;b$; by BZB$ in 1100
Insert:
20 VL=54296:AD=54277:SR=AD+1:WF=AD-1:
NO=17:N1=NO:LF=AD-5:HF=LF+1
25 POKE AD,255:POKE SR,48:POKE VL,15
50 C$(1)=CHR$(31):C$(2)=CHR$(28):C$(3)=
CHR$(30):C$(4)=CHR$(158)
60 P(1)=51:P(2)=34:P(3)=64:P(4)=38
90 DN$=CHR$(17):FOR K=1 TO 5:
DN$=DN$+DN$:NEXT K:
DN$=CHR$(19)+DN$
1010 PRINT C$(A);
1015 P=(A-1)*9+3:Z$=RIGHT$(STR$(A),1)
1130 PRINT CHR$(144):RETURN
2010 SD=15:SP=4:N1=33:GOSUB 7000:RETURN
2510 SD=10:SP=10:N1=33:GOSUB 7000:RETURN
7000 REM *** BEEP SD,SP
7010 POKE VL,15:POKE WF,N1
7020 POKE LF,SP:POKE HF,SP:
FOR DD=1 TO SD*50:NEXT DD
7030 POKE HF,0:POKE LF,0:N1=NO:RETURN

BBC Micro

Replace AT Y,X by TAB(X,Y). For example, line 6010 becomes:
6010 PRINT TAB(1,20);S$;TAB(1,20)
Replace INKEY$ by INKEY$(0)
Insert:
20 MODE 2
25 COLOUR 135:CLS
60 C(1)=1:C(2)=2:C(3)=4:C(4)=5
80 B$=CHR$(35)
1010 COLOUR C(A)
1015 P=(A-1)*4
1070 SOUND 1,-10,P(A),40/(N+1):FOR DE=1
TO 2000/N:NEXT
1130 COLOUR 0:RETURN
2010 SOUND 1,-15,2,40
2510 SOUND 1,-15,40,40

PLAN OF ACTION

Our examination of programming techniques has so far concentrated on documentation, and the need to make each section of a program clear and understandable. Here we look at the wider implications of program design and consider the questions that should be asked before any code is written.

Program design is seen by those involved — the designer/programmer and the user — as a grand and formal exercise in applied problem-solving. Unfortunately, the problems to be solved are always assumed to be of the technical programming kind — how to format the screen, how to make this loop faster, where to fit everything into RAM, and so on — whereas the real problems are present from the start of the project, and are usually created at the first meeting of the user and the 'expert'. Users are rarely very clear about the true nature of their problems — they hope the expert will tell them what the problem is and how to solve it — and experts very often think they know the problem and the solution before the user even begins to state it. The result is bad initial communication, leading to an incomplete description of the problem and the user's requirements. Working from this specification is bound to produce an unsatisfactory system that the user may be pressurised into accepting.

For home computing projects the programmer is usually also the designer (or 'systems analyst') and the consumer. This should mean that communication problems are lessened considerably. Nonetheless, as a combined user/designer, you should always make the effort to explain problems, solutions and requirements to yourself as clearly as if you were talking to another person.

Let's consider an imaginary user and his problem: he's a keen aircraft modeller who also owns a cassette-based microcomputer. He wants to store fairly detailed descriptions of materials used in the construction of each model that he makes so that when working on later models he can search his records for previous use of this kind of glue, or that type of joint. What the designer must therefore get from the user is a clear statement of the following:

■ The program function. This can start off as a vague statement of intent such as 'It should store my model records', but it must be refined by the designer's persuasion and interrogation into something more like a requirement specification, such as 'It should store my descriptions of the model and its construction and materials, as typed in at the keyboard, and display them when I type in the model's name, or some aspect of its construction.' This states the user's needs rather more clearly, and points to some of the specific programming tasks involved (storing, searching, indexing, retrieving, etc.).

■ How the program will be used. Some of the physical details of typical usage may be clear from the function description, but these may not be complete. For example, the user may not want the model details displayed on the screen because he works in a shed without a television set. In this case, a 'hard copy' print-out of selected details may be required.

■ What it will look like — input and output formats. The professional programmer will often use pre-printed charts representing the screen to draw each display that the user will see during input/output phases. Such elaborations are not often necessary for home use, although high resolution graphics may be an exception to this. Screen formats are a very important aspect of the *user interface* — the interactions between user and machine — and deserve the sort of close attention and discussion that is sometimes given to more obviously ergonomic aspects of computing, such as the positioning of keyboards and monitors, height of the table, and levels of illumination, etc.

■ How it should be organised — file and program formats. The user may feel that he needs to store at least 100 aircraft descriptions, and anything less will be useless. On the other hand, he may only ever build half a dozen more or less standard models. The size of a program's data files has serious implications for their format and access methods. A serial scan through six model descriptions on cassette taking, say, five minutes may be quite acceptable to the user, whereas waiting for 100 to be searched would be out of the question. A solution might be to put the program and description index file on one tape, and the descriptions themselves on 20 other tapes classified by aircraft type, for example.

The size of the program itself can also become a problem: if the text input section requires a complex text editor, if the program is fat with menus and heavy with significant messages, if the file-handling sections employ complicated searching and indexing routines, then the program may have to be split into several separate programs in order to fit into the available RAM.

■ What it should do — special procedures and calculations. In the model aircraft example, these

are unlikely to arise but they often do so when other problems are considered. There may be 20 perfectly good and equivalent ways of going through a particular process but the user may well insist on one and only one of these.

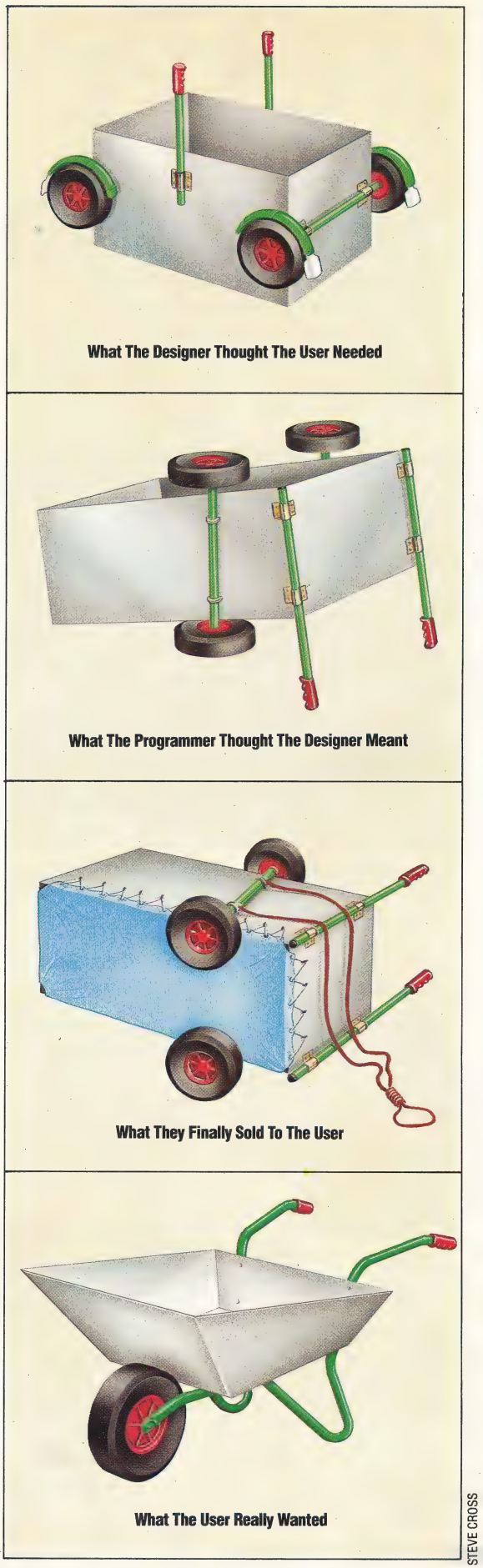
Getting this wrong makes the user immediately dissatisfied with the program. The designer may be tempted away from the user's preferred method by the greater efficiency of other methods, but any advantage is quickly lost if the user won't use the program! Discovering the user's procedures can be very helpful when it comes to designing calculations. Why invent a formula for calculating wing-loadings, for example, if you can simply ask the model maker how he does it?

With all this information noted down, the job of translating the specification into a program can begin. A useful approach to take is to design the user-program dialogue first, then the data files and then the processes that control it all. The word 'dialogue' is taken to mean the two-way communication of information that goes on between the user and the program. This does not simply consist of the input of model aircraft details and their subsequent display but also includes every prompt, message and menu that the program produces and every input, command or selection that the user enters. It is also important to fix the style of dialogue at this stage. For the aircraft program a choice between menu and command-driven interactions might be appropriate. The decisions taken here will have a considerable effect on the structure of the overall program. The contents and format of the dialogue must be considered in detail, but the reward for this effort is that all the data manipulated by the program should now be specified. This means that the storage space required for error messages and prompts can be calculated and, most importantly, the files can now be designed.

For the model aircraft program, where files will contain large blocks of text and will be very lengthy, splitting the file up onto several tapes so that each can be searched more quickly may be the best solution. If it warrants the effort, the data may be compressed by a coding algorithm before it is written to tape, and then decoded on reading.

By this time, the necessary functions will be apparent. There will be routines to allow the data text to be added and edited, to file the newly input text (these should update any indexes used by the system), to accept component names, to search for and display descriptions, etc. All these must be presented as options to the user, and they must all be able to deal with invalid data.

At this stage it is advisable for the user to make a careful check on the design to ensure that it performs as it should. If all is well, the program can now be coded. Of course, this is easier said than done, and the act of turning the design into an efficient working program may well reveal further problems.



What The Designer Thought The User Needed

What The Programmer Thought The Designer Meant

What They Finally Sold To The User

What The User Really Wanted

STEVE CROSS

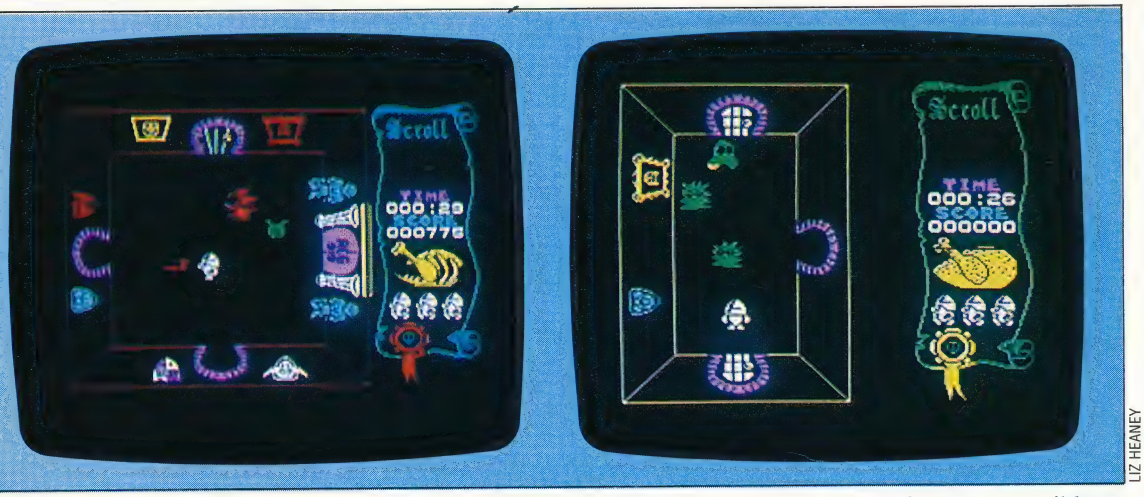
Describe, Define, Design
If the typical software development team of user, designer and programmer had set out to solve the problem of moving heavy loads around gardens, this is what they might have produced. Bad communication — between expert and non-expert, and amongst experts — is still a major problem facing all design teams

GHOSTS IN THE MACHINE

Ultimate, Play The Game is a company that has gained a well-deserved reputation for producing high-class computer games. Early releases from Ultimate set new standards in Spectrum animated graphics. Here we look at Atic Atac, a game that adds the strategy elements of adventure gaming to the fast action of the arcade.

Atic Antics

These screen photographs show two stages in Atic Atac — an adventure game that takes place at the frantic pace of an arcade game. A succession of monsters and other surprises appear as you hunt through corridors and secret passages in search of a golden key



LIZ HEANEY

Atic Atac is one of the rare breed of games that manages to combine the excitement of arcade action with the complexity of the adventure game. In classic adventure style, the game is set in the many rooms of a haunted castle. You are trapped inside and can escape only by finding the golden key that unlocks the main doors. Your life is threatened by a collection of evil creatures — spiders, ghouls, monks, witches, devils and hungry monsters are all here, to say nothing of Dracula, Frankenstein's Monster, the Mummy and lashings of bats. It is rather like being in an over-populated Hammer Horror movie. Trapdoors and hidden passages abound, and there are various useful or valuable objects for you to collect.

So far it all sounds much like a typical adventure game, but it is illustrated by superb animated graphics, which is where the arcade aspect of Atic Atac comes in. For you are presented with a three-dimensional coloured plan view of each room or dungeon. You can see the various doors leading off north, south, east or west, and other trappings include suits of armour, bookshelves, grandfather clocks and pictures. In your chosen guise of Knight, Wizard or Serf (or, as the program would have it, 'surf') you move through each room, armed with the appropriate weapon for your character. The various monsters keep appearing, heralded by puffs of smoke, and their every touch weakens you badly. Fortunately your armament of axe, sword or spell will repel your attackers. Most of the monsters appear to move fairly randomly, but some have a deliberately antisocial policy of heading straight for you.

Movement is controlled by a Kempston or cursor joystick or the keyboard. Unfortunately, like a lot of games, Atic Atac uses the keys Q,W,E and R for movement, and as these are in a straight line on the keyboard the game is unnecessarily difficult to play. It would be far more sensible to use Q and A for up and down, and alternative keys on the bottom row for left and right.

The game could easily become a simple shoot-out if it wasn't for the fact that some of the objects, not to mention the golden key itself, are very useful. Some coloured doors will open only if you possess the matching coloured key, and certain objects ward off certain creatures. On a more basic level you need to find food, as the rapidly disintegrating chicken to the right of the screen vividly portrays. If you don't eat, it slowly changes from a wholesome fowl to a heap of bones, signifying your death by starvation.

Initially, Atic Atac may be considered as simply a very frustrating arcade game. Once you begin to master the techniques of fighting the various monsters, you begin to appreciate the adventure side of it as you search the castle for various artefacts and treasure. But be warned — this is a game that requires many hours of effort before the key is found and the doors are opened.

Atic Atac: for the 48K Spectrum, £5.50
Publishers: Ashby Computers and Graphics Ltd., Ashby de la Zouch, Leicestershire LE6 5JU
Authors: Ultimate, Play The Game
Joysticks: Kempston and cursor joysticks
Format: Cassette

ROUTINE SHAPE-UP

Sprite graphics assist in creating fast-action arcade-style games in BASIC. Machines like the Commodore 64 have chips dedicated to controlling sprite shapes; other machines do not possess this hardware but are capable of supporting sprites using software. We look at a routine to create and move a sprite on the BBC Micro.

Sprites can take many shapes and forms, but all must be designed on a common grid. There is no restriction on the choice of grid dimensions, but it is most sensible to make the grid a whole number of bytes wide (i.e. 8, 16, 24 bits etc.) and to choose a depth that gives a broad rectangular or square workspace. The routine we give uses a grid 24 pixels wide by 21 pixels deep. 63 bytes of memory will therefore be used to hold the sprite's shape. The shape is defined by designing a shape on the grid and then coding that design into binary. Here, each pixel that we wish to have turned on in the final shape is coded as a one and each pixel that is to be turned off is coded as a zero. Once the bytes making up the shape have been defined as binary values, they must then be converted to either decimal or hex, and placed in an area of memory. The design of the demonstration sprite is shown.

The 63 numbers that define the sprite can be entered as DATA statements and READ by the BASIC part of the program. As each number is READ in, it must be placed in an area of memory specially set aside for the purpose. This area can be anywhere in RAM as long as it is protected so that it cannot be overwritten during a program run. The most obvious place to store the sprite data is at the top of the BASIC program area. The top of this area is defined by the variable HIMEM. So that our data is not overwritten, it is first necessary to lower HIMEM a little. Lines 220 and 230 of the program do this and set the address of the first byte of data, SPRDAT, to start just above the new value of HIMEM. Lines 1740 to 1770 read the data and place it in the 63 bytes starting at location SPRDAT.

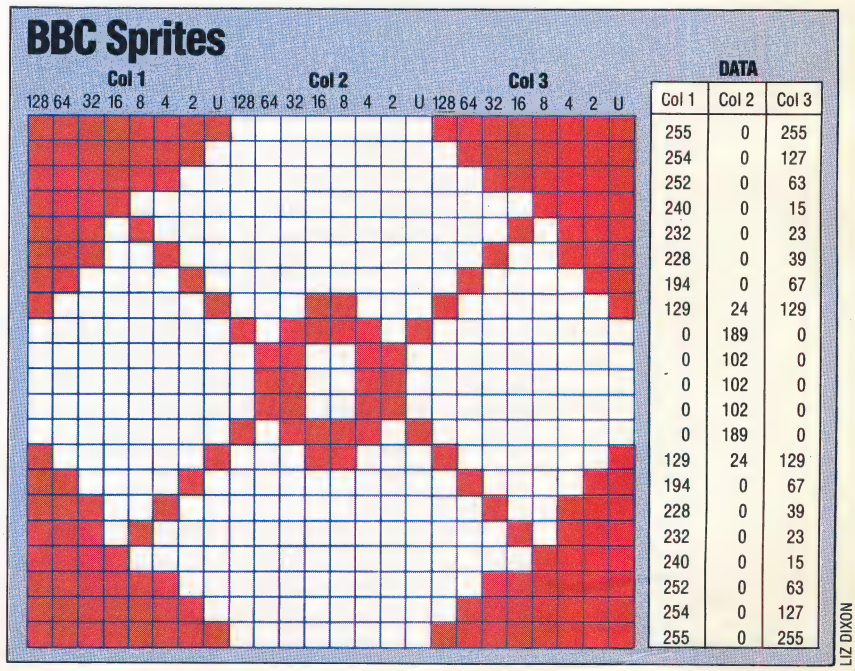
THE MACHINE CODE ROUTINE

The machine code's main function is to analyse the sprite design and then to carry out the operations necessary to convert the data into a screen display. To do this, the machine code routine must look at each bit of the 63 bytes of data in turn and for each bit decide whether to plot a point (if the bit is one) or leave a space (if the bit is zero). Probably the easiest way of analysing each bit of a particular byte is to use one of the Rotate instructions. Line 1100 of the source code uses the ROtate Left (ROL) instructions on a particular byte. This instruction causes each bit in the byte to move one place to the left. The old value of the carry flag is inserted at the right-hand end, and the bit that 'falls off' the left-hand end is shifted into the carry. By repeated ROLs, the routine can examine each bit in turn, whilst it is in the carry. On any subsequent ROL, the bit will be returned to the right-hand end of the byte. As long as we are careful not to alter the carry flag between ROLs, then the byte will have returned to its original value after nine rotations.

	C								
Original Contents	C	1	1	0	1	1	1	0	0
After 1st ROL	1	1	0	1	1	1	0	0	C
After 2nd ROL	1	0	1	1	1	0	0	C	1
After 3rd ROL	0	1	1	1	0	0	C	1	1
After 4th ROL	1	1	1	0	0	C	1	1	0
After 5th ROL	1	1	0	0	C	1	1	0	1
After 6th ROL	1	0	0	C	1	1	0	1	1
After 7th ROL	0	0	C	1	1	0	1	1	1
After 8th ROL	0	C	1	1	0	1	1	1	0
After 9th ROL	C	1	1	0	1	1	1	0	0

You can see from this demonstration that the original contents of the carry flag (C) do not particularly concern us, as this is rotated through the byte and eventually shifted out of the byte

Sizing The Sprite
The sprite measures 24 × 21 pixels, and so maps onto 63 bytes, one pixel per bit in the usual hi-res method. The colour of the whole sprite is set by the variable, logcol, while the sprite size is determined by the two scale factors, XSCALE and YSCALE; these should be even numbers



back into the carry flag. Notice that the ROL in line 1100 is used in the absolute indexed addressing mode so that each of the 63 sprite bytes in turn is accessed and analysed in this way.

Once the routine has isolated a particular bit and decided whether to plot a point or a space, the job of actually plotting to the screen has to be done. There are two methods of doing this on the BBC Micro. The first method is to POKE values directly into the area of memory that controls the screen display. On the BBC, this is not as easy as it sounds. Two main problems occur. Firstly, the screen memory areas vary between model A and model B machines, and between modes. Secondly, the mathematical relationship between pixels and bits in screen memory is very complex. For example, in mode 2 each byte of memory controls only two pixels on the screen. As this mode supports 16 colours, each pixel needs four bits to define its colour. In mode 0, however, a two-colour mode, each pixel requires only one bit for definition. Designing a routine to code our sprite format into mode 2 format would be feasible, but the routine would then only work in mode 2.

Fortunately, there is an alternative method. When dealing with graphics in BASIC, the BBC operating system has to do the sort of manipulations that we require. We can access this operating system routine to do most of this difficult work for us. In addition, a routine written using this operating system call will work in all the graphics modes. The routine works in the same way as a VDU command in BBC BASIC. For example, to plot a single point on the screen, the following VDU command could be used:

```
VDU25,68,300;700;
```

Here, the x and y co-ordinates are specified by the numbers 300 and 700 respectively. This VDU command can be duplicated in machine code by using the operating system call OSWRCH. This call is made repeatedly after first placing a number in the accumulator. Because the accumulator can only hold one byte at a time, the x and y co-ordinates must be split into LO-byte/HI-byte form as follows:

```
VDU25,68,44,1,188,2
```

To perform this command in machine code, OSWRCH needs to be called six times. The vector to the start address of OSWRCH is held in location &FFEE, and the routine is accessed by JSR &FFEE. Any VDU command can be done in this way, and this routine uses OSWRCH calls in several places. Note that whereas OSWRCH does not affect the values of the X, Y and A registers, it will alter the contents of the carry flag. Hence, when preservation of the carry is required, the contents of the carry must be stored away before OSWRCH is called. This is the case with the ROL routine. The easiest method of preserving the carry is to push the processor status register onto the stack (PHP) before calling OSWRCH, and pull it off again afterwards (PLP).

Let's now look at the structure of the machine code routine. The main analysis of sprite data and plotting to the screen are performed using the subroutine SPRPLT, starting at line 890 in the source code. The first operation of this routine is to set the method of plotting as an Exclusive OR operation. This is entirely similar to the GCOL command in BBC BASIC. Points plotted in this way can be erased by simply replotting over the top. Any screen data under the sprite will therefore be left intact. At the start of the machine code an absolute move is made to position the top left hand corner of the sprite. Each row is then analysed by taking three bytes of sprite data and rotating them as described earlier.

A relative plot or relative move is made over a distance determined by a horizontal scaling factor, XSCALE, depending on the value of the data bit currently in the carry. At the end of each row of three bytes an absolute move is again made, to the same x co-ordinate as the sprite's top left corner, but to a reduced y co-ordinate determined by a vertical scaling factor, YSCALE. The process is repeated until all 63 bytes have been analysed.

The SPRPLT subroutine is used in two places. Firstly, to erase the old sprite by replotting over it, and secondly to plot the new sprite. After plotting the new sprite, its co-ordinates are transferred to OLDX and OLDY in preparation for the next time the routine is used.

Using The Machine Code Routine From BASIC

The routine can be easily used by BASIC programmers without needing to understand the routine itself. The steps that must be carried out are as follows:

1) Design your sprite and place the data in an area of memory as shown in the program
2) Set the display mode you wish to use
3) Set the values of XSCALE and YSCALE as shown in line 1870
4) Set the logical colour for the sprite as shown in line 1890
5) Set the x and y co-ordinates of the position you wish the sprite to appear in and use the procedure given at lines 2010 to 2060 to convert absolute co-ordinates into LO-byte/HI-byte form.
6) CALL SPRITE

The machine code routine can be incorporated into your BASIC listing, as here. The assembler listing can be suppressed by changing line 260 to:

```
FOR opt%=0TO2STEP2
```

Alternatively, you can save the routine once assembled (i.e. after a run) using the *SAVE command, taking careful note of the start and end addresses of the code given as the assembly listing is displayed.

Erratum

The program segments given in the boxes at the bottom of page 295 for Inserting and Deleting a record in a BASIC array were inadvertently transposed. Lines 100-190 are used for inserting a record, and lines 200-250 for deleting a record from the array. The program lines and captions are otherwise correct

BBC Sprite

```
10 REM **** BBC SPRITES ****
20 :
30 REM ** SET UP ZERO PAGE VARIABLES **
40 TYPE   =&70
50 OLDXLO =&71:?OLDXLO=0
60 OLDXHI =&72:?OLDXHI=0
70 OLDYLO =&73:?OLDYLO=0
80 OLDYHI =&74:?OLDYHI=0
90 NEWXLO =&75
100 NEWXHI =&76
110 NEWYLO =&77
120 NEWYHI =&78
130 XSCALE =&79
140 YSCALE =&7A
150 logcol =&7B
160 ROW    =&7C
170 YTMPLO =&7D
180 YTMPHI =&7E
190 XTMPLO =&7F
200 XTMPHI =&80
210 :
220 HIMEM=HIMEM-150
230 SPRDAT =HIMEM+1
240 OSWRCH =&FFEE
250 DIM MC% &01FF
260 FOR opt% =0TO 3 STEP 3
270   P% =MC%
280   [
290   OPT opt%
300   \ **** MOVE TO OLD X,Y ****
310   \
320   .SPRITE  LDA #25
330            JSR OSWRCH
340            LDA #68
350            JSR OSWRCH
360            LDA OLDXLO
370            STA XTMPLO
380            JSR OSWRCH
390            LDA OLDXHI
400            STA XTMPHI
410            JSR OSWRCH
420            LDA OLDYLO
430            STA YTMPLO
440            JSR OSWRCH
450            LDA OLDYHI
460            STA YTMPHI
470            JSR OSWRCH
480   \
490   \ **** RUBOUT OLD SPRITE ****
500   \
510            JSR SPRPLOT
520   \
530   \ **** MOVE TO NEW X,Y ****
540   .NEWMOV   LDA #25
550            JSR OSWRCH
560            LDA #68
570            JSR OSWRCH
580            LDA NEWXLO
590            STA XTMPLO
600            JSR OSWRCH
610            LDA NEWXHI
620            STA XTMPHI
630            JSR OSWRCH
640            LDA NEWYLO
650            STA YTMPLO
660            JSR OSWRCH
670            LDA NEWYHI
680            STA YTMPHI
690            JSR OSWRCH
700   \
710   \ **** PLOT NEW SPRITE ****
720   \
730            JSR SPRPLOT
740   \
750   \ **** TRANSFER NEW X,Y TO OLD X,Y ****
760            LDA NEWXLO
770            STA OLDXLO
780            LDA NEWXHI
790            STA OLDXHI
800            LDA NEWYLO
810            STA OLDYLO
820            LDA NEWYHI
830            STA OLDYHI
840   \
850   \ **** RETURN TO BASIC ****
860   \
870            RTS
880   \
890   \ **** SPRITE PLOTTING SUBROUTINE ****
900   \
910   \ ** SET EXCLUSIVE OR PLOT **
920   .SPRPLOT LDA #18
930            JSR OSWRCH
940            LDA #3
950            JSR OSWRCH
960            LDA logcol
970            JSR OSWRCH
980   \
990   \
1000  \ ** INITIALISE COUNTS **
1010  \ ** X COUNTS BYTES, Y COUNTS BITS**
1020  \ ** ROW COUNTS ROWS OF THREE BYTES **
1030           LDX #&00
1040  .NEWROW LDA #&00
1050           STA ROW
1060  \
1070  .BYTE    LDY #&09
1080  .BIT     LDA #&65
1090           STA TYPE
1100           ROL SPRDAT,X
1110           PHP          \STORE CARRY ON STACK
1120           BCS DOPLOT
1130           LDA #64
1140           STA TYPE
1150   \ ** VDU PLOT COMMAND **
1160   .DOPLOT LDA #25
1170           JSR OSWRCH
1180           LDA TYPE
1190           JSR OSWRCH
1200           LDA XSCALE
1210           JSR OSWRCH
1220           LDA #&00
1230           JSR OSWRCH
1240           JSR OSWRCH
1250           JSR OSWRCH
1260   \ ** END OF PLOT COMMAND **
1270           PLP          \RETRIEVE CARRY
1280           DEY
1290           BNE BIT
1300   \ ** IF BYTE FINISHED **
1310           INX
1320           CPX #63
1330           BEQ FINISH
1340   \ ** CHECK FOR END OF ROW **
1350           INC ROW
1360           LDA ROW
1370           CMP #3
1380           BNE BYTE
1390   \ ** IF END OF ROW SUBTRACT YSCALE FROM Y **
1400   \
1410           LDA YTMPLO
1420           SEC
1430           SBC YSCALE
1440           STA YTMPLO
1450           BCS NOSUB
1460           DEC YTMPHI
1470   \ ** ABSOLUTE MOVE TO START OF NEXT ROW **
1480   \
1490   .NOSUB  LDA #25
1500           JSR OSWRCH
1510           LDA #68
1520           JSR OSWRCH
1530           LDA XTMPLO
1540           JSR OSWRCH
1550           LDA XTMPHI
1560           JSR OSWRCH
1570           LDA YTMPLO
1580           JSR OSWRCH
1590           LDA YTMPHI
1600           JSR OSWRCH
1610   \
1620   \ ** NEXT ROW **
1630           JMP NEWROW
1640   \
1650   \ ** END OF SUBROUTINE **
1660   .FINISH RTS
1670   \
1680 ]
1690 NEXT
1700 :
1710 REM **** BASIC PROGRAM STARTS HERE ****
1720 :
1730 REM ** READ SPRITE DATA **
1740 FOR address = SPRDAT TO SPRDAT+62
1750   READ data:?address = data
1760 NEXT address
1770 :
1780 REM **** SET M/C PARAMETERS ****
1790 :
1800 MODE1
1810 GCOL0,129
1820 CLG
1860 ?XSCALE =4:?YSCALE =4
1870 ?logcol=1
1880 :
1890 X=700:Y=800
1900 PROCCOORDS (X,Y)
1910 CALL SPRITE
1920 :
1930 REM **** WAIT FOR CRSR KEYS ****
1940 FOR S=0 TO 1 STEP 0
1950 PROCKEYS
1960 PROCCOORDS (X,Y)
1970 CALL SPRITE
1980 NEXT S
1990 END
2000 :
2010 DEF PROCCOORDS (X,Y)
2020 XH=X DIV 256:XL=X MOD 256
2030 YH=Y DIV 256:YL=Y MOD 256
2040 ?NEWXLO=XL:?NEWXHI=XH
2050 ?NEWYLO=YL:?NEWYHI=YH
2060 ENDPROC
2070 REM **** SCAN KEYBOARD ****
2080 DEF PROCKEYS
2090 LOCAL LT,ZZ:LT=2
2100 FOR ZZ=0 TO 1 STEP 0
2110 IF INKEY(-58) THEN Y=Y+50:ZZ=LT
2120 IF INKEY(-42) THEN Y=Y-50:ZZ=LT
2130 IF INKEY(-26) THEN X=X-50:ZZ=LT
2140 IF INKEY(-122) THEN X=X+50:ZZ=LT
2150 NEXT ZZ
2160 ENDPROC
2170 REM **** SPRITE DATA ****
2180 DATA 255,0,255,254,0,127,252,0,63
2190 DATA 240,0,15,232,0,23,228,0,39
2200 DATA 194,0,67,129,24,129,0,189,0
2210 DATA 0,102,0
2220 DATA 0,102,0
2230 DATA 0,102,0
2240 DATA 0,189,0,129,24,129,194,0,67
2250 DATA 228,0,39,232,0,23,240,0,15
2260 DATA 252,0,63,254,0,127,255,0,255
```

Spritely Does It

The sprite — defined in the 63 bytes of DATA — can be moved around the screen by the arrow keys. Its relative slowness is the consequence of using the OSWRCH ROM routine, but it will work in all modes. When the program is RUN, the assembly listing scrolls the screen first, followed by the graphic display

LLAMA SOFT

Llamasoft is a company that is built around the talents of one man — Jeff Minter. Minter has achieved star status in the software world with his quirky, humorous games for Commodore and Atari machines. Here we look at his career to date, and examine his obsession with camels, llamas, sheep and other animals.

Most software houses are started in 'cottage industry' fashion, with one or two programmers working from home. As they become more successful, other programmers are hired and the very factors that led to success become submerged in the rush to produce new software; the 'house style' disappears and one company's output becomes much like any other's. But Llamasoft is different — Revenge of the Mutant Camels and Sheep In Space could not have come from any other company.

Llamasoft's distinctive style must be attributed to one man. Jeff Minter is still only 22 years old but has, over the past few years, become one of the stars of the software world. It is his obsession with camels, llamas and other furry beasts that has shaped Llamasoft's image, and his favourite adjective — 'awesome!' — is used regularly in Llamasoft advertising. Minter's fame — or notoriety — is such that he is lampooned in a rival company's software: one of the hazards in Software Projects' Manic Miner is described as 'Attack of the Mutant Telephones'.

Minter began writing games in 1981 after leaving the University of East Anglia, where he studied maths and physics. Despite coming top of his set in the computer part of his course, he failed the maths papers and was forced to leave.

His first venture into the commercial software market came when he wrote a version of Defender for the Vic-20 and founded Llamasoft in 1982 to market the product. Traxx and Gridrunner soon followed, and success was assured. Gridrunner, in particular, sold well, both in Britain and in the USA. Minter's mother Hazel, who helps to run Llamasoft, claims: 'Gridrunner was the top seller in America for ages and ages. That was the one that really made Jeff's name'.

Minter now writes games for the Vic-20, Commodore 64 and Atari computers, but is not interested in producing games for the Spectrum, which he considers an 'awful machine'. Conversions of Llamasoft games have therefore been undertaken by Salamander Software and Quicksilva, which produce the Spectrum versions under licence. Llamasoft will shortly hire an assistant to convert games, thereby allowing Minter to concentrate on writing new software. But Minter will remain totally in control of the programming. He explains: 'I'm just interested in getting a good game. They're all my ideas, and so there's no reason to take on other people'.

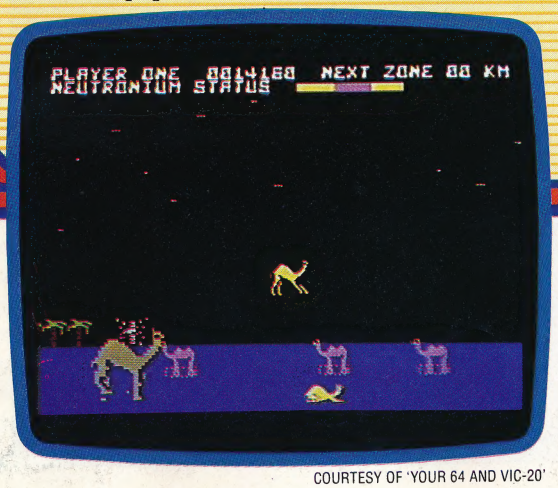
The company is very much a family concern. Llamasoft became a limited company in April 1984, and its directors are Minter himself, his father Patrick, and his mother Hazel. The company is run from the family home in Tadley, Hampshire. Patrick Minter helps his son with the programming, while Hazel handles the administration. Llamasoft also employs two assistants, two accountants and an artist.

Minter has now produced 21 games for Llamasoft; many of these contain a recurring theme, usually camel-oriented. Minter says he is 'deeply into camels', and the first game to feature these beasts was written in 1982 for the Atari. This was Attack of the Mutant Camels, and it was later followed by Revenge of the Mutant Camels.

Llamasoft has now signed a contract with CBS to market Minter's games in the UK (with the exception of the Quicksilva and Salamander conversions). Distribution in the United States is handled by Human Engineering Software. Demand is high for Llamasoft games: a new product will have a first run of 10,000 cassettes, and this run is then repeated a number of times, depending on sales.

Minter does not foresee any major changes in the way the company is run. He claims that the business side does not interest him, and he plans to carry on as before, developing his obsession with furry animals and writing 'awesome' new games.

TONY LODGE



COURTESY OF 'YOUR 64 AND VIC-20'

Revenge Of The Mutant Camels
Llamasoft's surrealist-obsessive style is graphically illustrated in this shot from Revenge Of The Mutant Camels — follow-up to the successful Attack Of The Mutant Camels

Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

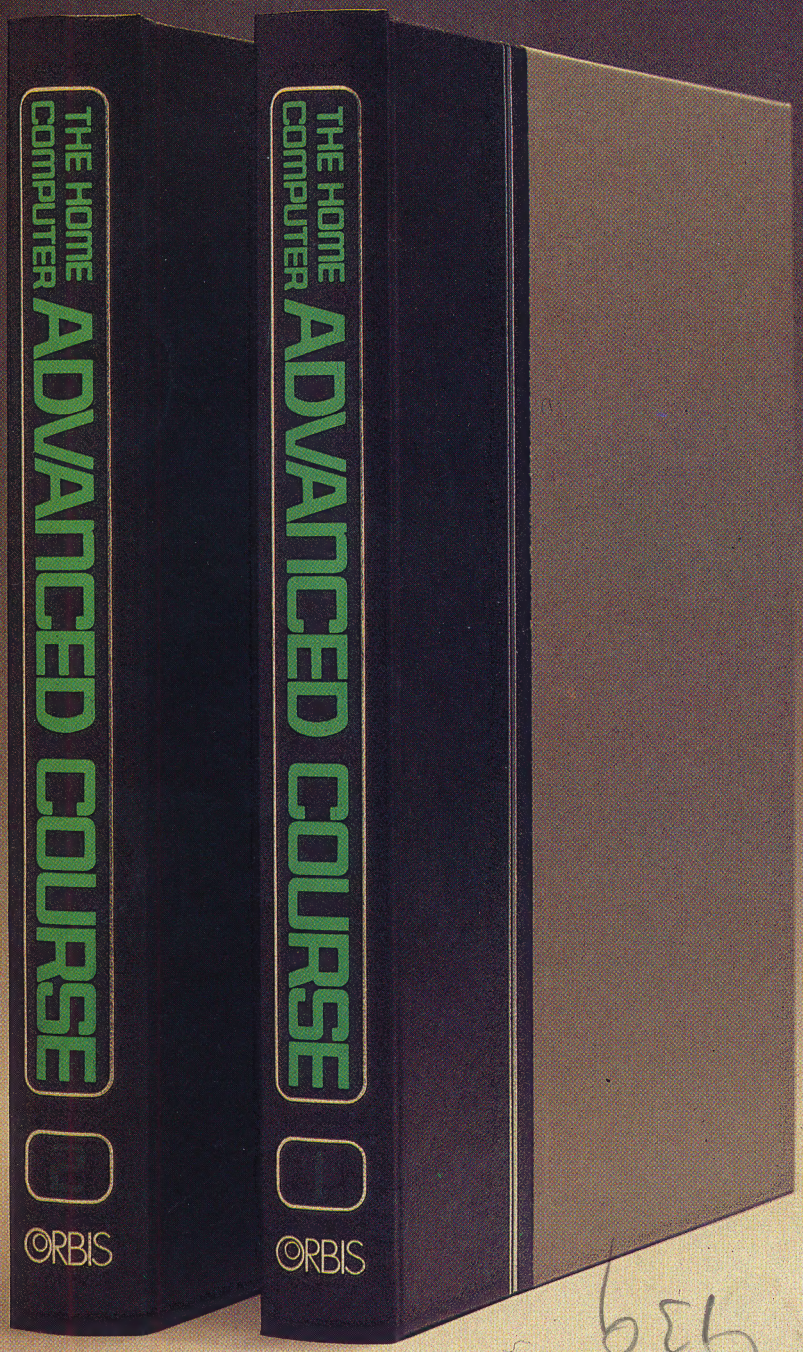
And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

THE HOME COMPUTER ADVANCED COURSE

WE HAVE DESIGNED BINDERS SPECIALLY TO KEEP YOUR COPIES OF THE 'ADVANCED COURSE' IN GOOD ORDER.



All you have to do is complete the reply-paid order form opposite – tick the box and post the card today – **no stamp necessary!**

By choosing a standing order, you will be sent the first volume free along with the second binder for £3.95. The invoice for this amount will be with the binder. We will then send you your binders every twelve weeks – as you need them.

Important: This offer is open only whilst stocks last and only one free binder may be sent to each purchaser who places a Standing Order. Please allow 28 days for delivery.

Overseas readers: This free binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain binders now. For details please see inside the front cover. Binders may be subject to import duty and/or local tax.

The Orbis Guarantee: If you are not entirely satisfied you may return the binder(s) to us within 14 days and cancel your Standing Order. You are then under no obligation to pay and no further binders will be sent except upon request.

PLACE A STANDING ORDER TODAY.